**Лабораторна робота №4**

**Память ПК. Код исправления ошибок**

**Цель работы**: изучение особенностей функционирования памяти и реализации кода с исправлением ошибок

*Короткие теоретические сведения*

Память компьютера из-за всплесков напряжения и по другим причинам время от времени может ошибаться. Чтобы бороться с ошибками, используются специальные коды, умеющие обнаруживать и исправлять ошибки. В этом случае к каждому слову в памяти особым образом добавляются дополнительные биты. Когда слово считывается из памяти, эти дополнительные биты проверяются, что и позволяет обнаруживать ошибки.

Помехозащитные коды делятся на две группы:

1. Код с обнаружением ошибок

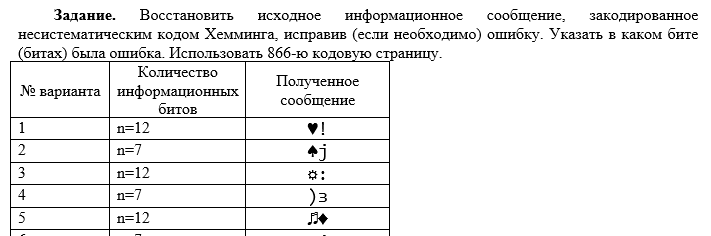
2. Код с исправлением ошибок

Код с исправлением ошибок – самокорректирующийся код. Код Хэмминга

Коды, в которых возможно автоматическое исправление ошибок, называются самокорректирующимися. Для построения самокорректирующегося кода, рассчитанного на исправление одиночных ошибок, одного контрольного разряда недостаточно. Для того чтобы можно было исправлять ошибки, кодовая комбинация должна обладать некоторой избыточностью, которая достигается за счет добавления нескольких контрольных разрядов.

Предположим, что слово состоит из m бит данных, к которым мы дополнительно прибавляем r бит (контрольных разрядов). Тогда общая длина слова составит n бит (то есть n = m + r). Единицу из n бит, содержащую m бит данных и r контрольных разрядов, называют кодовым словом.

**Завдання**



**Рішення**

**Текст програми:**

#include <iostream>

#include <windows.h>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

// Получает строку данных. Возвращает строку с исправленными ошибками

// wordSize - размер кодовых слов

string decode(string str, int wordSize = 12);

// Получает вектор закодированных битов и возвращает

// Массив исправленных информационых битов

// wordSize - размер кодовых слов

vector<bool> decode(vector<bool> code, int wordSize = 12);

// Преобразует переданную строку в массив битов

vector<bool> getCode(string str);

// Преобразует массив битов в строку

string getStrOfCode(vector<bool> code);

// Получает кодовое слово и возвращает разшифрованное

vector<bool> decodeHamming(vector<bool> word);

// Преобразует массив битов в число

int toNumFromWord(vector<bool> word);

// Преобразует переданное число в вектор битов

// size - разрядность числа

// (то есть переводит число из 10 СЧ в 2

vector<bool> toWord(int val, int size = 32);

// Преобразует слово в виде строки в слово в виде битов

vector<bool> toWord(string strWord);

// Преобразует слово в виде битов в слово в виде строки

string toStrWord(vector<bool> word);

/\*\*

\* Запрашивает от пользователя значение нужного типа

\* @param{char[]} label - текст, что будет показанно пользователю

\*

\* Пример работы:

\* prompt<int>("Введите целое число: ");

\* prompt<char>("Введите символ: ");

\* prompt<string>("Введите строку: ");

\*/

template <typename T>

T prompt(const char label[]);

void printHello();

int main() {

   printHello();

   while (true) {

      int wordSize = prompt<int>("Enter the size of the code word: ");

      string input = prompt<string>("Enter the Message: ");

      cout << endl;

      string output = decode(input, wordSize);

      cout << " Input: " << input << endl;

      cout << "Output: " << output << endl << endl;

      if (prompt<string>("Again? (1 - yes): ") != "1") break;

      cout << "- - - - - -" << endl << endl;

   }

   return 0;

}

string decode(string str, int wordSize) {

   vector<bool> code = getCode(str);

   return getStrOfCode(decode(code, wordSize));

}

vector<bool> decode(vector<bool> code, int wordSize) {

   if (wordSize < 3) wordSize = 3;

   vector<bool> decode;

   for (int i = code.size(); i >= wordSize; i -= wordSize) {

      vector<bool> word(code.begin() + i - wordSize, code.begin() + i);

      vector<bool> wordDecode = decodeHamming(word);

      decode.insert(decode.begin(), wordDecode.begin(), wordDecode.end());

   }

   return decode;

}

vector<bool> getCode(string str) {

   const short BITS\_IN\_CHAR = 8;

   const int len = str.length();

   const int size = len \* BITS\_IN\_CHAR;

   vector<bool> code(size);

   for (int i = 0; i < len; i++) {

      auto word = toWord(int(str[i]), BITS\_IN\_CHAR);

      for (int j = 0; j < BITS\_IN\_CHAR; j++) {

         code[i \* BITS\_IN\_CHAR + j] = word[j];

      }

   }

   return code;

}

string getStrOfCode(vector<bool> code) {

   const short BITS\_IN\_CHAR = 8;

   const int size = code.size();

   string str;

   for (int i = 0; i <= size - BITS\_IN\_CHAR; i += BITS\_IN\_CHAR) {

      vector<bool> charWord(code.begin() + i, code.begin() + i + BITS\_IN\_CHAR);

      str += (char)toNumFromWord(charWord);

   }

   return str;

}

vector<bool> toWord(int val, int size) {

   vector<bool> word(size, 0);

   for (int bit = 0; bit < size; bit++) {

      word[size - bit - 1] = val & (1 << bit);

   }

   return word;

};

int toNumFromWord(vector<bool> word) {

   int size = word.size();

   int res = 0;

   for (int bit = size - 1; bit >= 0; bit--) {

      res = res \* 2 + word[size - bit - 1];

   }

   return res;

};

vector<bool> decodeHamming(vector<bool> word) {

   int size = word.size();

   if (size < 3) {

      cerr << "Code word have to have least 3 bits" << endl;

      return word;

   }

   short errors = 0; // Количество несовпавших проверочных битов

   int errIndex = -1; // Индекс бита с ошибкой

   for (int r = 1; r <= size; r <<= 1) {

      int res = 0;

      for (int i = r; i <= size; i += 2 \* r) {

         for (int j = 0; j < r; j++) {

            int index = i + j - 1;

            if (index >= size) break;

            res ^= (int)word[index];

         }

      }

      if (res != 0) {

         errors++;

         errIndex += r;

      }

   }

   if (errors >= 2) {

      if (errIndex >= size) return word; //Code word has more then 1 errors

      word[errIndex] = !word[errIndex];

   }

   //else if (errors == 1) cout << "Error in parity bit by index";

   vector<bool> res;

   for (int i = 1; i <= size; i++) {

      if ((i & (i - 1)) == 0) continue; //Проверочный бит

      res.push\_back(word[i - 1]);

   }

   return res;

}

vector<bool> toWord(string strWord) {

   vector<bool> word;

   for (char strBit: strWord) {

      word.push\_back(strBit == '1' ? 1 : 0);

   };

   return word;

}

string toStrWord(vector<bool> word) {

   string strWord;

   for (bool bit: word) {

      strWord += (bit == 1) ? "1" : "0";

   };

   return strWord;

}

void printHello() {

   cout << "\* \* \* Hamming decode \* \* \*" << endl << endl;

}

template <typename T>

T prompt(const char label[]) {

   cout << label;

   while (true) {

      T val;

      cin >> val;

      if (cin.fail()) {

         cin.clear();

         cin.ignore(32767, '\n');

         cout << "Wrong. Try again: ";

      }

      else {

         cin.ignore(32767, '\n');

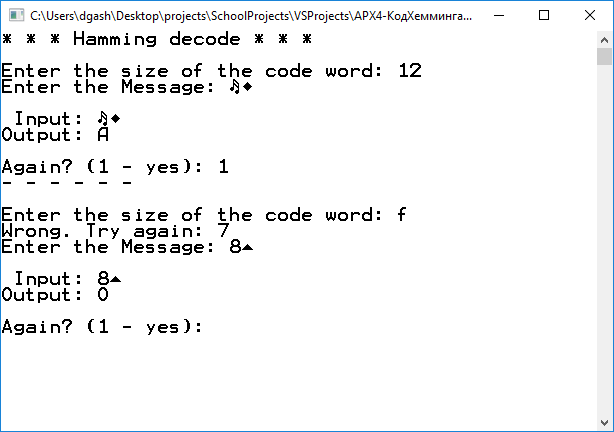
         return val;

      }

   }

}

**Результат выполнения:**



**Висновок:** на этой лабораторной работе я изучил особенности функционирования памяти и реализации кода с исправлением ошибок.